ソフトウェア工学　136－4
（２００２．３．７）

オブジェクト指向日本語一貫記述環境 OOJDE の開発

永松泰成　　畠山正行

茨城大学工学部情報工学科

ドメインユーザ向けのプログラム開発環境としてオブジェクト指向日本語一貫記述環境 OOJDE を開発した．本環境は従来から研究を進めてきた日本語一貫プログラミング環境の流れを汲むものであり，プログラム開発における要求分析からプログラム生成(分析，設計，実装)までを複数の相似な記述を行うものである．OOJDE で扱う言語はすでに開発されているオブジェクト指向自然日本語 OONJ，オブジェクト指向設計記述言語 OOPD であり，各言語記述を支援する環境をそれぞれ構築することで，上流工程からのスムーズな記述作業を実現した．一貫した相似な記述であることを検証する機能として各段階の対応関係表示機能と相互関連・相互作用連動表示機能を特に本環境に実装した．これにより各段階の相似性の比較・検証が可能となり，統合環境として有用性が高まった．

Development of Object-oriented Japanese Description Environment OOJDE

Yasunari Nagamatsu　　Masayuki Hatakeyama

Ibaraki University

Object-oriented Japanese description environment OOJDE has been developed as a program development environment for the domain users. This environment has been realized to describe from the requirement analysis up to the program generation in system development using the Natural Japanese. The languages treated in OOJDE is OONJ (Object-oriented Natural Japanese) and OOPD(Object-oriented Program Design description Japanese). The development of the OONJ environment and the OOPD environment have individually been performed. Then the smooth description operations throughout the developing process was realized. To realize an integrated support environment, two special functions were implemented,(1)The function to show the correspondence among each stage. (2) The function to show the mutual relationship and the mutual interaction among each stage.Then, the similarity among the developing processes have been attained.

1　はじめに

我々は科学技術計算などの分野のドメインユーザ向けに日本語でプログラムを記述する開発環境を研究している．すでに日本語一貫プログラミング環境 JOMON[1]が開発されているが，JOMON で扱う言語であるオブジェクト指向記述日本語 OODJ[4]は，日本語プログラム言語 JSMDL[1]とのギャップが大きく，記述とその変換の間の一貫性という意味で不十分であった．そこで一貫性を強めて言語記述ギャップを少なくするため，分析，設計段階における言語仕様の改善が行われ，新たに日本語一貫記述を実現する言語として定めたのがオブジェクト指向自然日本語 OONJ[2]，オブジェクト指向設計記述言語 OOPD[3]である．OONJ は OODJ を包含しプログラム開発の工程において分析段階に相当する．OOPD は分析段階とのギャップを埋めるために従来の JSMDL をより上流工程に移し，ユーザの可読性を高めた実行可能（プログラム言語 C++へトランスレートできる）な設計段階の言語として設計されている．この OONJ と OOPD に自然日本語を加えた言語仕様を総称してオブジェクト指向日本語 OOJ

(Object-Oriented Japanese)と呼称する.

従来のシステム開発環境では分析，設計等の各段階において異なった言語，ツール等を使うために各段階での変換が正しく行われているかなどを検証することは困難であった. それに対して OOJ では一貫性を備えた言語系を用いるために各段階での変換プロセスは明確化できるようになると考えられるので，各段階での比較，検証が可能と考えられる．そこでその特性を強化するため多段階変換記述間での対応関係等を表示する機能等をつけ，記述するユーザに変換前後の記述を直接比較させ，記述ミスや不足等を明示的に把握させることで記述の信頼性を向上させる事を試みる．そのため具体的には分析，設計，実装の各段階における変換記述の相似性の検証に繋がる機能として，各言語間での対応する部分を表示する機能と各段階で相互関連先，相互作用先を表示したときに他段階で対応するものがあれば連動して表示するような機能を環境に追加する。

本研究では従来より高度な日本語一貫記述を実現するために OONJ, OOPD それぞれの環境構築を行い，オブジェクト指向日本語記述（OONJ，OOPD）によってプグラム開発における分析・設計記述を実現し，設計記述からトランスレータによってプログラムを自動生成でき、各段階間での相似性が検証できる統合環境を構築する。

2　オブジェクト指向日本語一貫記述環境 OOJDE

2.1　OOJDE の言語仕様とその位置付け

OOJDE 環境で扱う言語は以下の４つである.

- 自然日本語（NJ）
- オブジェクト指向自然日本語（OONJ）
- オブジェクト指向設計記述言語（OOPD）
- オブジェクト指向プログラミング言語（OOPL,C++）

各言語の役割りを図１に示す.

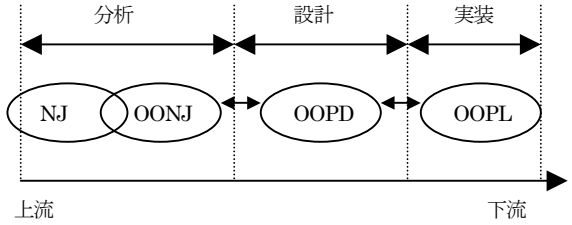

図１　各言語仕様の位置付け

2.2　OOJDE の構成

OOJDE は以下の４つの環境を含み，各環境間の記述を比較できる機能を付けた統合環境である.

１．NJ（自然日本語）記述エディタ
２．OONJ 記述エディタ
３．OOPD 記述エディタ
４．OOPL 記述エディタ

基本的な環境使用の流れは，まずユーザは対象世界の分析をはじめ NJ 記述エディタに対象世界を日本語文章で書き出していく. そして OONJ 記述エディタにおいて日本語文章にオブジェクト指向の制約を設けた記述を行っていく．OONJ 記述を行った後 OOPD 記述エディタで設計を行う. OOPD 記述が完成すれば OOPD トランスレータによってソースコードが生成される.

システム開発を行う上で，各段階で正しく変換できているかどうか比較をするために OOJDE では一貫相似性検証機能として次のような機能を設ける.

- **対応関係表示機能**
- **相互関連・相互作用先の連動表示機能**

上記機能の詳細は５章で述べる.

3　OONJ 記述環境

3.1　OONJ の特徴とその記述形式

オブジェクト指向自然日本語 OONJ は自然日本語にオブジェクト指向の特性を付加した言語である. OONJ ではオブジェクト指向記述構成要素の種類とその構造的な記述構成要素の全てをリスト形式で提示する（表 1）. 表１の構成要素種類の番号と種類名をファセット番号と呼び，これによって一意特定識別を実現する．OONJ の詳細な構文規則等については[2]を参照されたい.

表１　オブジェクト指向記述モデルの記述要素「種類」リスト

対象記述モデル
1.1　モノ名　1.2　属性名　1.3　振舞い名
2.1　参照属性
2.2.1　汎化属性　2.2.2　集約属性　2.2.3　一般関連属性
2.3　相互作用属性
2.4.1　一般制約属性　2.4.2　モノ制約属性
2.4.2　属性制約属性　2.4.4.　振舞い制約属性
3.1　内部振舞い　3.2　相互作用伝達　3.3.　相互作用
4.0　駆動記述モデル
5.0　シナリオ名

3.2　OONJ 記述環境の設計

分析記述段階では，対象世界の自然日本語記述からオブジェクト指向の特性であるモノ，属性，振舞いの抽出をフレーム単位で行うことになり，そのフレームの階層的構成で対象世界を示す．また，各フレームの内容はスロットの積み重ねになるが，スロットは OONJ 記述において“＃”で区別が可能になる．そこで，OONJ 記述環境のデータ構造は OONJ フレーム記述を単位として，各フレームがそのフレーム名，フレーム番号，関連フレーム，フレーム内容（スロットの集合）を持つ構造にした[2].

3.3　OONJ 記述環境の構成とその機能

実際に構築した OONJ 記述環境使用例が図 2 である．環境内で用いたサンプルは一貫してライフゲームである。それらの一部を付録として付けている．

・　**自然日本語（NJ）記述**

図 2 中の中央上部のウィンドウが NJ 記述を行うエディタである．ユーザは対象世界の概念記述を日本語文章で行うことになる．

・　**OONJ 記述**

ユーザは NJ 記述を参照しながら OONJ 記述をフレーム単位で行っていく．フレームはスロットの集合になるが”#”でスロット単位を示す．記述をわかりやすくするために予約語（ファセット番号など）や定義済みフレーム番号等を色分け表示をする．またポップアップメニューに OONJ 記述構成要素リスト（表 1 参照）を追加し，ユーザの情報整理を支援し，選択したファセット番号をエディタ上に書き出しユーザを支援する．

・　**別フレーム，全フレーム表示**

OONJ 記述ウィンドウでは定義済みのフレーム番号は赤色で表示されている．その赤色のフレーム番号をマウスで選択し，ポップアップメニューから「別フレーム表示」を選択すると図 2 中の右下のような小さなウィンドウが表示され，その選択したフレーム内容が表示される．またフレーム全表示ボタンを押すことで，図 2 中の右のようなウィンドウが表示され，そのウィンドウ内に全てのフレーム内容が表示される．

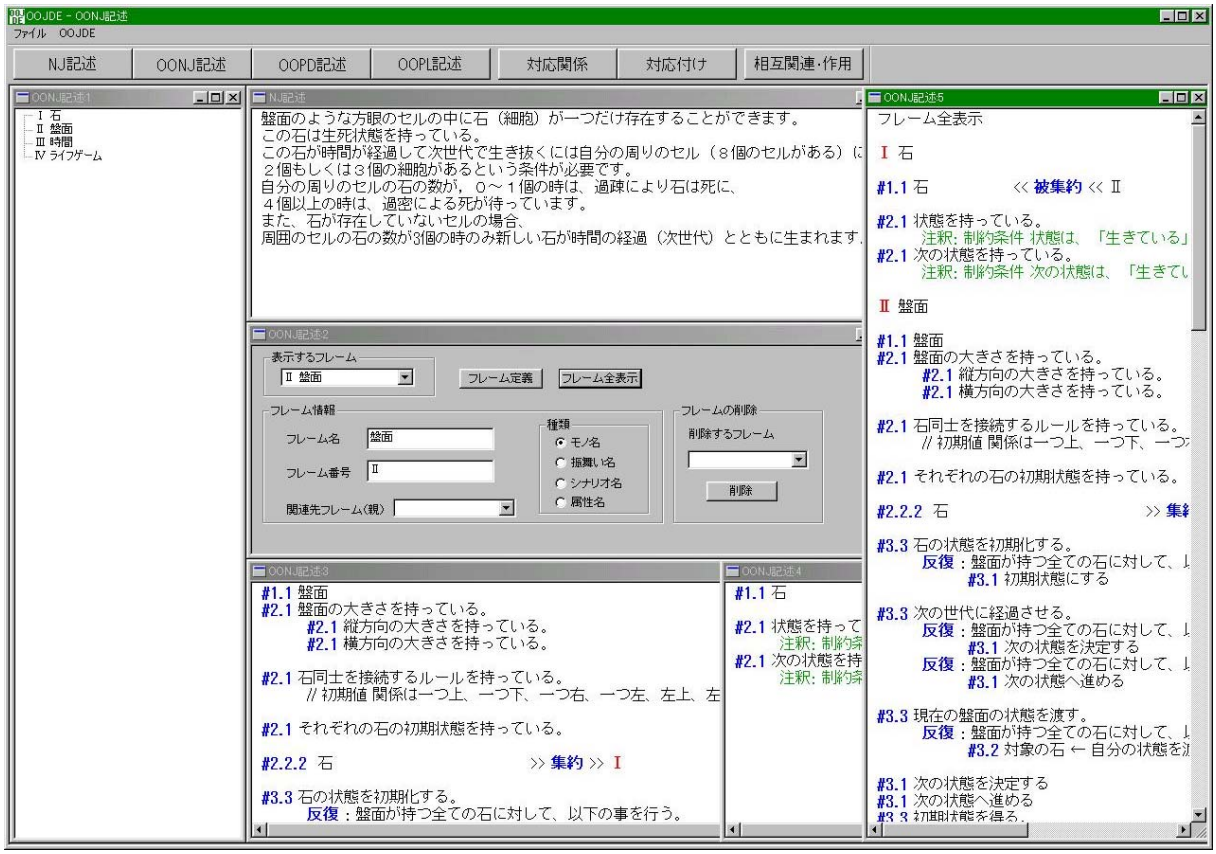

図 2　OONJ 記述環境

4　OOPD 記述環境

4.1　OOPD の特徴

OOPD はプログラム設計記述を目的とし，トランスレータによって実行可能なプログラムへ変換する記述言語である． OOPD は全体の振舞い記述部分と個々のオブジェクト定義部分とに記述を明確に分け，メッセージ送信形式などに明示的に変換することで一意性を持ち実行可能なプログラムへ変換ができるようになる．OOPD の詳細な記述モデル，構文規則に関しては[3]を参照されたい．

4.2　OOPD 記述環境の設計

設計記述段階では，対象世界の自然日本語記述あるいは分析段階の OONJ 記述からクラス，クラスの属性，クラスの動作（メソッド），クラス間関係を定義することになり，クラス定義記述を外部仕様と内部仕様に分け，定義したクラスのインスタンスを生成し使用する操作系記述を行うことになる．OOPD 記述のデータ管理としては OOPD クラス記述と OOPD 操作記述を単位として行い，各クラス記述がそのクラスの属性リスト，動作リスト，クラス関係リストを持つ構造になっている．

4.3　OOPD 記述環境の構成と機能

構築した OOPD 記述環境の使用例を図 3 に示す．OOPD は詳細設計段階であるので，図 3 からわかるように複数の入力フォームを持った様々なダイアログを用意し，ユーザはその入力フォームに必要事項だけを記入していく形式を取り，できる限りユーザの入力を少なくし，かつ詳細な事項を正確に決められるようにしている．ダイアログの種類としては「クラス定義」「属性定義」「動作定義」「操作定義」などがありユーザはダイアログからクラスを生成していく．クラスを生成するとツリービューにクラスが登録されツリーアイテムの外部仕様・内部仕様をそれぞれ選択することでエディタ上に外部仕様･内部仕様記述を展開する．OOPD にはトランスレータ[1] [3]があり指定したフォルダに C＋＋のソースコードを生成することになる．クラスは”cls”，属性は”attribute”などの通し番号でソースコードを生成する．辞書作成ダイアログを利用することで，生成するソースコードに適切な英字名を付けることも可能である．

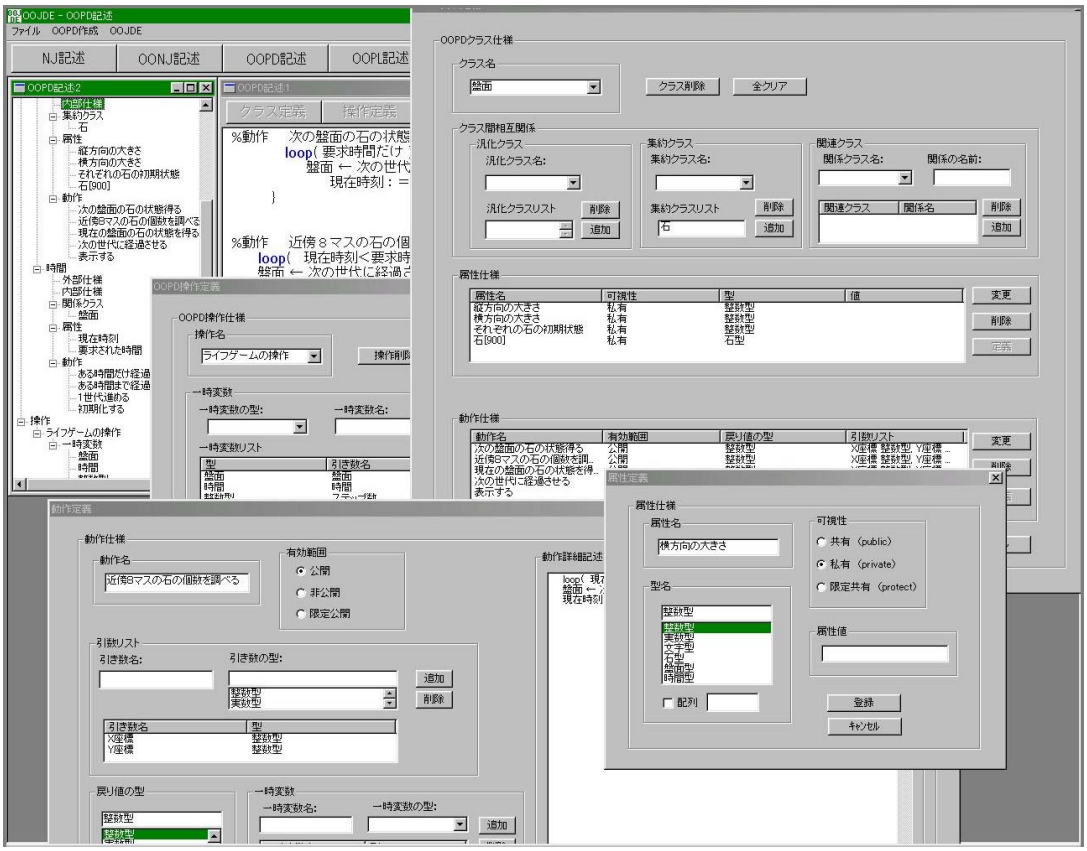

図 3　OOPD 記述環境

5　統合環境上の一貫相似性検証機能の実現

5.1　対応関係表示機能

5.1.1　対応関係表示機能

OOJDE 記述環境における対応関係表示機能は分析（OONJ），設計（OOPD），実装（OOPL）の各段階についての記述の対応する範囲をマウス等で指定し，その対応すべき別段階の記述を指定する．ユーザが指定したものが対応付けられ，対応関係が記述環境内に蓄積される．ユーザが再びある段階での記述を指定し対応関係の表示を行うことで，それに対応する他の段階の記述を探し，その部分を反転表示するというものである．

対応関係は１対１になるとは限らないので，複数結びついている場合は順次表示していく方針をとる．

対応関係でシステムに蓄積したデータは各環境のファイルとは別形式の拡張子（*.jde）のファイルに保存する．

5.1.2　対応関係表示機能の実現と操作

まず「対応関係」を選択することで OONJ 記述，OOPD 記述，OOPL 記述の各ウィンドウを図４のように移動し３つ並べて表示する．

・　対応付け

ユーザは OONJ 記述，OOPD 記述，OOPL 記述の各ウィンドウでマウス選択により対応付けたい範囲を指定する．そして，この対応付けボタンを押すと各ウィンドウで選択された部分が対応付けられたことになり，環境内にストックされる．

・　対応間係の各操作

OONJ，OOPD，OOPL の各環境のポップアップメニューには「対応関係表示」「対応関係の次表示」「対応関係の変更」「対応関係の削除」の４項目がある．ユーザはこれらのメニューを使用しながら対応関係表示等の操作を行う．

1．対応関係の表示

OONJ，OOPD，OOPL の各記述ウィンドウのいずれかのウィンドウにおいて対応関係が付いている部分を選択し，メニューから「対応関係表示」を選択することで他段階のウィンドウにおいて対応付いている部分があればその部分を図 4 のように反転表示する．

２．対応関係の次表示

「対応関係の次表示」メニューを選択すると「対応関係表示」メニューで行った表示にさらに対応するものがあればそちらに反転表示を切替える．

３．対応関係の変更

「対応関係の変更」を選択すると「対応関係表示」or「対応関係次表示」で直前に表示した対応関係に変更を加える．変更を加える記述はポップアップメニューを呼び出したウィンドウのみである．

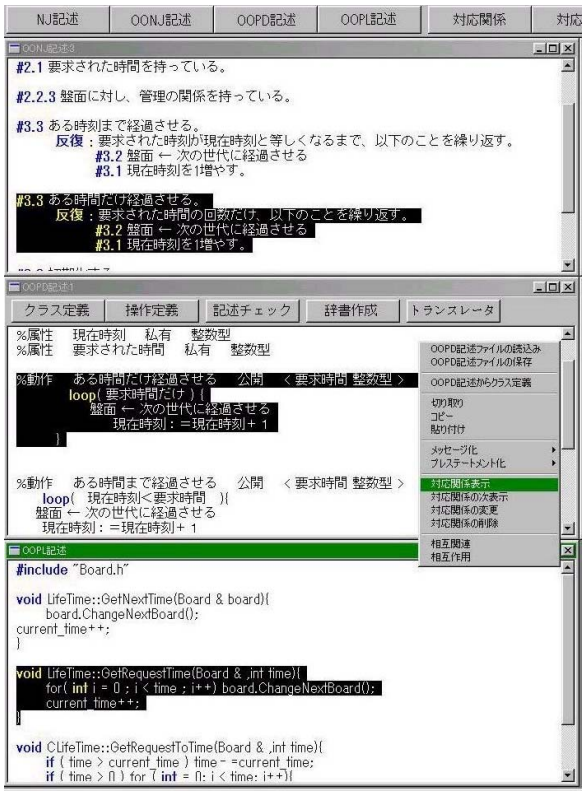

図４　対応関係表示画面

5.2　相互関連・作用先の連動表示機能

5.2.1 各段階の相互関連･相互作用先の連動表示機能

OONJ 環境，OOPD 環境，OOPL 環境は各々相互関連先（汎化，集約），相互作用先を表示する機能がある． OONJ，OOPD，OOPL の各エディタが同様な機能を持ち合わせているがこの機能を３つのエディタで連動させることで相似性の検証に有用になると考えられる．具体的にはある段階で相互関連先もしくは相互作用先を参照する（関連先，作用先のフレーム（OONJ），クラス（OOPD，OOPL）の

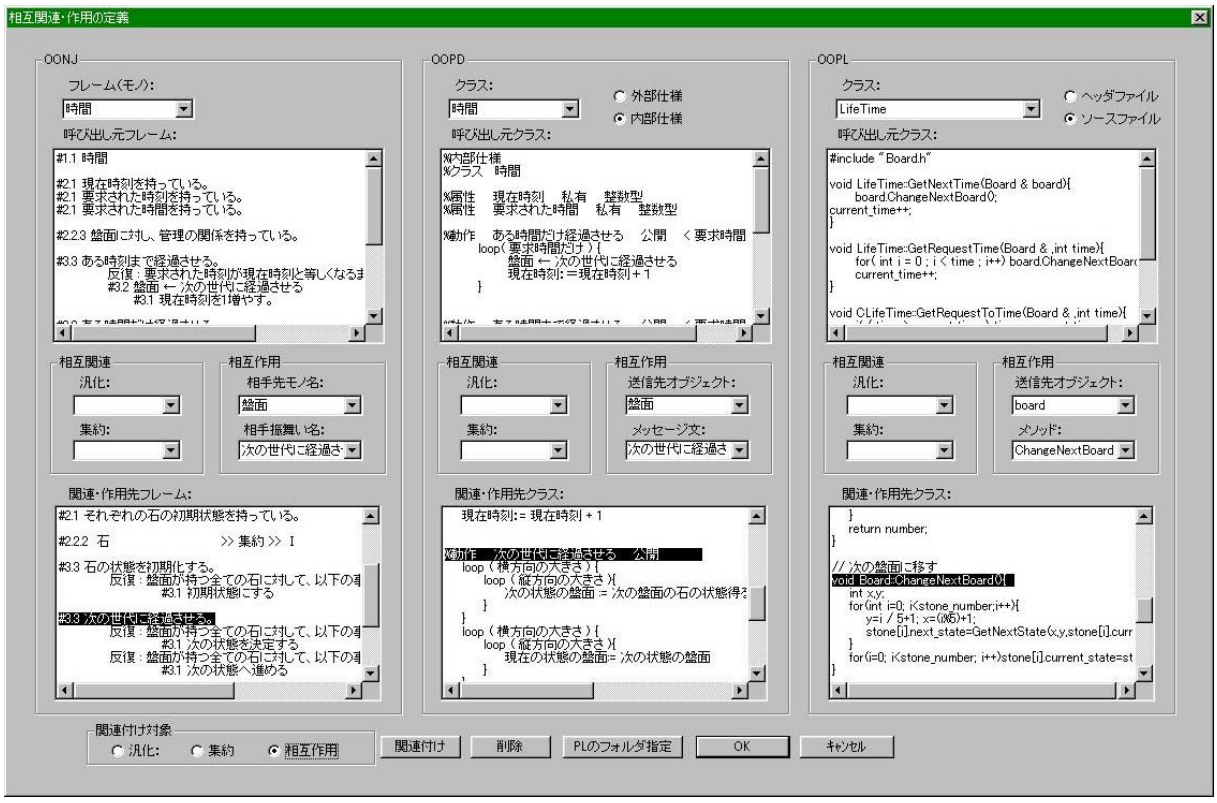

図 5　相互関連・相互作用関連付けダイアログ

記述に切替える）と他の段階の記述も関連先もしくは作用先の内容に切り替るといったものである．

5.2.2 相互関連，相互作用の連動表示機能の実現

・　各段階の相互関連･相互作用

OONJ, OOPD に関しては図 5 のダイアログ起動と供に OONJ（フレーム），OOPD（クラス）コンボボックスにデータが自動的に入力される．OOPL に関してはファイルが一つではないので対象世界を示すフォルダを用意しシステム側でファイルの解析を行いクラスの抽出をする．

上段のエディットボックスにフレーム内容，クラス内容が展開されると，その展開された内容から相互関連の場合，「汎化」「集約」コンボボックスに該当するフレーム，クラスが追加され，相互作用の場合「相手先モノ名」「送信先オブジェクト」コンボボックスに該当するモノ，オブジェクトが追加されていく．相互関連の場合は「集約」「汎化」を指定することで下段のエディットボックスにその内容を表示する．相互作用の場合はさらに相手モノ名に対するメッセージが追加されることになり，そのメッセージを選択することで下段のエディットボックス内で検索を行い，該当するメッセージ名を選択表示することになる．

・　相互関連・相互作用の関連付け

図 5 のダイアログにおいて OONJ, OOPD, OOPL 各々で相互関連もしくは相互作用の表示を行った後で，図 5 中左下の「汎化」「集約」「相互作用」のラジオボタンの中からいずれかを選択し「関連付け」ボタンを押すことで OONJ，OOPD，OOPL の間で関連情報が付けられ，そのリンク内容が環境に蓄積される．

・　連動表示

相互関連・相互作用関連付けダイアログを利用して関連付けると，各エディタで相互関連・相互作用先を参照したときに他の段階の記述エディタも切り替る．例えば図 5 の相互作用関係を関連付けているとする．図 6 は相互作用前の各エディタの状態である．ここで OOPD 記述ウィンドウにおいて「盤面←次の世代に経過させる」という部分をマウスで選択し，さらにポップアップメニューを表示し，その中から「相互作用」を選択すると図 7 のような状態に切り替わる．相互関連（汎化，集約）の場合も同様

な操作になる．

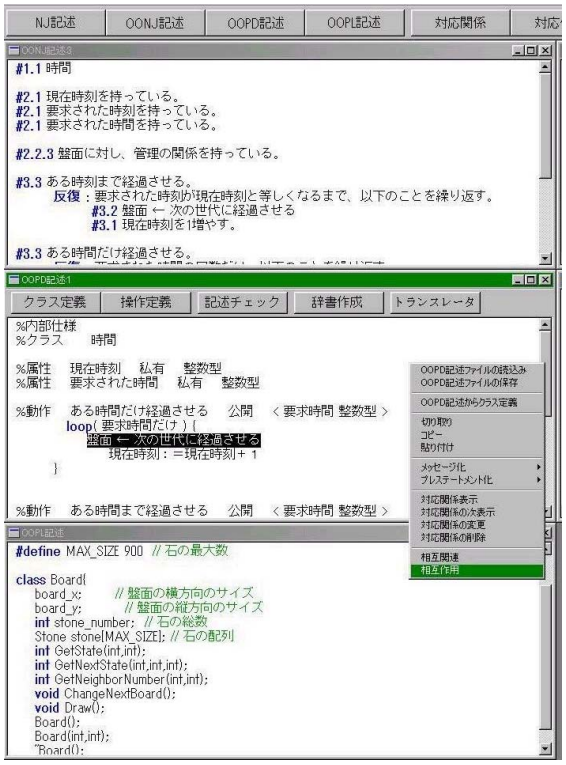

図６　相互作用前

図７　相互作用後

6　考察

6.1　OONJ 記述環境

OONJ 支援環境は NJ 記述からコピー&ペースト等での OONJ 記述作成が行えるので，一度 NJ 記述を行えば同じ文章をユーザは記述する必要はないので，複数ステップの言語仕様でも作業量等をストレスに感じることはないと考えられる．また，フレームの階層図の提示，フレームの全表示，OONJ 記述内での色分け表示等の機能はユーザの情報整理という意味で役に立つ機能であると考えられる．

6.2　OOPD 記述環境

OOPD 支援環境では入力フォーム形式の様々なダイアログを用意し，ユーザは入力フォームに属性名，動作名等を入力するだけで外部仕様，内部仕様の記述の枠組みを作り上げるので使いやすいものであると考える．また，内部仕様における動作の詳細記述に関してはポップアップメニューを利用したプレステートメント化等の機能によってユーザの作業負担を軽減することができた．

6.3　一貫相似性の検証機能

対応関係表示機能は記述するユーザに変換前後の記述の関係を直接比較させることで，それにより記述ミスや不足等が明示的に把握できるようになると考える．記述量の増加にともない対応関係を保持して表示できるという機能は，ユーザ自身がどのような分析，設計手順をたどったかなどを検証する上での一つの機能として位置付けられるだろう。

相互関連・相互作用先連動表示機能は分析，設計，実装の記述がオブジェクト指向パラダイムに沿った記述になっているかを判断する上において有効な機能といえる．そのことは図 5 のダイアログを見ても，各段階での記述の相似性を比較できることは明白である．また図６，図 7 のように３つのウィンドウを並べて比較することで分析記述を参照しながら設計記述を行ったり，設計記述から分析記述のあり方を見直すことが可能になり，従来からのトップダウン的な本プログラム開発環境にボトムアップ的な手法を実現したことにもなる．

7　結論と今後の課題

7.1　結論

システム開発において要求分析からプログラム生

成までをオブジェクト指向パラダイムを利用した一貫性を備えた日本語記述で行うことをサポートする支援環境としてオブジェクト指向日本語一貫記述環境 OOJDE を構築した．システム開発手順を対象世界の分析，設計，実装と明確に分け，それぞれの段階において言語仕様と環境を設けることでユーザは上流工程から段階的にスムーズな手順で作業を進めることが可能になった．また一貫相似性の検証機能によってユーザのプログラム開発を支援する環境として一応の機能を備えたものであるといえる．

7.2 今後の課題

現在の対応関係表示機能はユーザ主導で各段階の対応関係を付けるということで操作に手間がかかってしまい，かつ記述内容の変更に伴う対応関係の変更もユーザ側が行わなければならない。言語仕様間での対応関係が詳細な部分まで定義することができればシステム側で自動的に対応関係を付け，表示することも可能であると考えられ有効な機能に発展させることができるだろう．より多くのサンプルにこの機能を適応させることで有効性，問題点を検証をしていく必要がある．

参考文献

[1]加藤木和夫，畠山正行，「オブジェクト指向日本語一貫プログラミング環境」，情報処理学会論文誌，Vol.40，No.7，pp.3016-3030(1999)

[2]畠山正行，上田賀一「オブジェクト指向日本語 OONJ とその記述実験評価」，情報処理学会第 130 回ソフトウェア工学研究会研究報告，pp.153~160(2001)

[3]加藤木和夫，畠山正行，「オブジェクト指向プログラム設計記述言語 OOPD とその記述環境」，情報処理学会第 130 回ソフトウェア工学研究会研究報告，pp.161~168(2001)

[4]畠山正行，加藤木和夫，石井義之，「オブジェクト指向記述日本語 OODJ とその記述環境」，情報処理学会論文 Vol.41，No.9， pp. 2567 –2582 (2000)

付録　記述サンプル

```
フレーム番号　Ⅱ
#1.1 盤面
#2.1 縦方向の大きさを持っている.
#2.1 横方向の大きさを持っている.
#2.2.2　石                                    >> 集約 >>　I
#3.3 次の世代に経過させる.
  反復：盤面が持つ全ての石に対して，以下の事を行う.
    #3.1 石の次の状態を決定する
  反復：盤面が持つ全ての石に対して，以下の事を行う.
    #3.1 次の状態へ進める
```

```
フレーム番号 Ⅲ
#1.1 時間
#2.1 現在時刻を持っている.
#2.2.3 盤面に対し，管理の関係を持っている.
#3.3 ある時間だけ経過させる.
    反復：要求された時間の回数だけ，繰り返す.
        #3.2 盤面 ← 次の世代に経過させる
        #3.1 現在時刻を 1 増やす.
```

OONJ 記述例

```
%内部仕様
%クラス 盤面
%動作 次の世代に経過させる 公開
    loop ( 横方向の大きさ ) {
        loop ( 縦方向の大きさ ){
            次の状態の盤面 :=
        次の盤面の石の状態得る(x，y，現在の状態の盤面)
        }
    }
    ・・・
```

```
%内部仕様
%クラス 時間
%属性 現在時刻 整数型
%動作 ある時間だけ経過させる 公開 ＜要求時間 整数型
        loop( 要求時間だけ ) {
            盤面 ← 次の世代に経過させる
            現在時刻：＝現在時刻＋１
        }
```

OOPD 記述例

```
void Board::ChangeNextBoard(){
    int x，y;
    for(int i=0; i<stone_number;i++){
        y=i / board_x+1; x=(i%board_x)+1;
        stone[i].next_state=
        GetNextState(x，y，stone[i].current_state);
    }
    for(i=0;i<stone_number;i++)
    stone[i].current_state=stone[i].next_state;
}
```

```
void LifeTime::GetRequestTime(Board & , int time){
    for( int i = 0 ; i < time ; i++)
    board.ChangeNextBoard();
        current_time++;
```

OOPL 記述例